Doctor Dobbs Journal - CSPICE article.

CSPICE - A C Version of JPL's SPICELIB Toolkit

By

Ed Wright
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

What is SPICE?

The Navigation Ancillary Information Facility (NAIF), under the direction of NASA's Office of Space Science, built the SPICE data system to assist scientists with planning and interpretation of scientific observations from space borne-instruments. The system provides the ancillary information needed to recover the full value of science instrument data, and facilitate the correlation of individual instrument data sets with data from other instruments on the same or other spacecraft.

NAIF serves as the Ancillary Data Node of NASA's Planetary Data System, responsible for the distribution of the SPICE data sets, called kernel files, produced by NASA's planetary flight projects. The SPICE data kernels exist for:

S- Spacecraft trajectory, given as a function of time (SPK Kernels).

P- Planet, satellite, comet, asteroid, associated physical and cartographic constants (PCK Kernels).

I- Instrument information, including internal timing and other geometric information (I Kernels).

C- C matrix, time tagged orientation data of mounted structures and instruments (C Kernels).

E- Events for the spacecraft and ground data system, both planned and unplanned (E Kernels).

Hence the SPICE acronym. NAIF also assembles and distributes PCK kernels based on products provided by JPL's Solar System Dynamics Group.

The SPICE library (SPICELIB) consists of 952 portable FORTRAN routines with 79,369 lines of executable code and 153,649 comment lines. The library contains reader subroutines to retrieve data (position, velocity, and instrument observation geometry parameters) from each of the SPICE kernels, plus a wide assortment of geometry, math, time

Why Develop a C version of SPICE?

Many NAIF customers ask for a C version of the SPICELIB library. Not every site can access a FORTRAN compiler or programmer, but most possess a C compiler. C improves ease of use, and C libraries escape cross language I/O problems and non-portable interface issues. C integrates easily with Java, C++, and software environments such as IDL (Interactive Data Language).

What is CSPICE?

CSPICE extends the SPICE system to the C language realm. The functionality of CSPICE approximates that of SPICELIB, with minor differences due to the disparate properties of ANSI C versus FORTRAN. The CSPICE toolkit consists of an ANSI C version of the SPICELIB library, a support library, SPICE utility programs, documentation, and example cookbook programs. The toolkit's main component, the CSPICE library, includes the source for all C routines generated by f2c from SPICELIB routines, both f2c support libraries (libF77, libI77), a set of hand coded wrapper routines which encapsulate certain translated routines, and all required header files. Functional library ports exist for: HP, Solaris, MacPPC, SGI (o32 and n32), Win95/NT, Linux, and DEC Alpha Digital Unix.

Other C libraries exist which provide geometric/vector/matrix math functions (though not all are as numerically stable), but only CSPICE provides the ability to read and write SPICE kernel files. CSPICE also includes extensive time conversion routines and a sophisticated, user-configurable error trace/signal system which emulates exception handling.

SPICELIB to CSPICE

CSPICE'S existence requires the automatic conversion of FORTRAN code to C via the f2c utility. The utility creates C code which emulates the behavior of input FORTRAN code. The conversion of 79,000 plus lines of FORTRAN code to C is impractical without f2c, so lacking f2c, development of CSPICE would require a routine-by-routine rewrite of SPICELIB and take a horrendous amount of time.

NAIF uses a naming convention to distinguish between the various forms of a routine. Given a FORTRAN routine sub in file sub.f(or), f2c creates a C routine sub_ in file sub.c. The wrapper's name is sub_c in file sub_c.c and sub_c may call sub_. The code base of the CSPICE library is the set of all f2c translated routines.

The f2c application has several command line options. The CSPICE library builds with the options:

```
-u -C -a -A -P -!bs
```

```
-u    Make the default type of a variable "undefined."
-C    Compile code to check subscripts are within
      declared array bounds.
-a    Make local variables automatic instead of static
      unless they appear in DATA, EQUIVALENCE, NAMESLIST
      or SAVE statements.
-A    Produce ANSI C.
-P    Write a prototype file of ANSI or C++ for
      definitions in each input FORTRAN file.
-!bs  Do not recognize backslash escapes in character
      strings.
```

The Macintosh version of CSPICE does not use the -a argument due to a 32k size limit for local variables imposed by the MetroWerks CodeWarrior compiler.

The f2c distribution consists of the source for the application, as well as the source for the libI77 and libF77 libraries which simulate FORTRAN functionality in C. You may download the distribution from:

http://netlib.bell-lab.com/netlib/f2c/

Wrappers

The wrappers provide a C-friendlier interface to the more commonly used routines or to routines hand recoded in C and not dependent on an f2c translation (such as math functions). Compare the program interface between a FORTRAN routine, its f2c counterpart, and the corresponding wrapper. The FORTRAN version of SPKEZ has as an argument list:

```
SUBROUTINE SPKEZ ( TARG, ET, REF, ABCORR, OBS, STARG, LT )

INTEGER               TARG
DOUBLE PRECISION      ET
CHARACTER*(*)         REF
CHARACTER*(*)         ABCORR
INTEGER               OBS
DOUBLE PRECISION      STARG    ( 6 )
DOUBLE PRECISION      LT
```

where TARG, ET, REF, ABCORR and OBS are inputs, with STARG and LT the outputs. f2c creates an interface with the form:

```
int spkez_( integer    *targ    ,
            doublereal *et      ,
            char       *ref     ,
            char       *abcorr  ,
                    integer    *obs      ,
                    doublereal *starg    ,
                    doublereal *lt       ,
                    ftnlen      ref_len ,
                    ftnlen      abcorr_len )
```

Now the wrapper routine, which calls spkez_:

```
     void spkez_c ( SpiceInt             targ,
                    SpiceDouble          et,
                    ConstSpiceChar     * ref,
                    ConstSpiceChar     * abcorr,
                    SpiceInt             obs,
                    SpiceDouble          starg[6],
                    SpiceDouble        * lt       )
```

spkez_c passes C strings, single values (for input), and pointers (strings and outputs); spkez_ passes only FORTRAN strings, pointers (input and output), and the string length values. For those not experienced with C and FORTRAN, the differences between string formats may be unclear. Internally, f2c uses FORTRAN style stings: blank padded without null terminators.

Wrappers also replace system dependent calls. SPICELIB contains several routines with such calls; the most common return FORTRAN equivalents of argv, argc, and system time. f2c understandably fails to translate these routines to usable C, so those are manually recoded.

A number of non-portable SPICELIB routines provide some functionality available in the standard C library. The appropriate C calls replace the CSPICE versions of the routines either via a macro or wrapper.

The CSPICE design assumes calls to the library from pure C programs, not an f2c version of some program. A programmer should access CSPICE through a wrapper, assuming a wrapper for the needed routine exists. Otherwise direct calls to the f2c'd code base are needed. A wrapper may call other wrappers, but the f2c'd code base calls only other f2c'd code (sub1_ calls sub2_, not sub2_c).

Problems and Solutions

The process of creating a C version of a large FORTRAN library lends itself to numerous problems from code format and style issues to the use of internal data types. The requirement of full ANSI compatibility for the wrappers ensures few or no portability problems. Wrapper development uses the GNU C compiler (gcc) with the arguments -ansi (support ANSI standard), -Wall (warn of all errors), and -pedantic (reject non-ansi extensions).

The NAIF Team defined a coding standard for C routines which includes complete, informative, and human understandable internal documentation. NAIF SPICELIB routines contain extensive headers which list revisions, authors, platform specific modifications, as well as a detailed description of the routine's function.

With regards to generation of the wrappers, the original FORTRAN routines are the "seeds" for the wrappers. A simple perl script casts the FORTRAN comments to C style then creates a skeleton for the new subroutine containing the comments.

CSPICE and f2c use typedefs to emulate FORTRAN data types. CSPICE deliberately uses typedefs which differ from the f2c typedefs. The basic f2c typedefs:

```
     typedef long  int      integer;
     typedef short int      shortint;
     typedef float          real;
     typedef double         doublereal;
     typedef long  int      logical;
     typedef long  int      ftnlen;
```

f2c treats all characters and strings as char *. The most commonly used CSPICE typedefs:

```
     typedef char           SpiceChar;
     typedef double         SpiceDouble;
     typedef float          SpiceFloat;
     typedef long           SpiceInt;
     typedef const char     ConstSpiceChar;
     typedef const double   ConstSpiceDouble;
     typedef const float    ConstSpiceFloat;
     typedef const long     ConstSpiceInt;

     enum _Spiceboolean { SPICEFALSE = 0,    SPICETRUE = 1 };

     typedef        enum _Spiceboolean SpiceBoolean;
     typedef  const enum _Spiceboolean ConstSpiceBoolean;
```

The const types are not required by a need of function, but their use insures input values are not unintentionally modified within a routine. All functions with input-only vectors and matrices have those arguments declared constant.

A problem exists with the function declarations in stdlib.h on the Sun platform. The f2c header file, f2c.h, defines several macros which conflict with those found in stdlib.h. As some CSPICE functions require both f2c.h and stdlib.h our solution is to copy the needed typedefs to CSPICE instead of including f2c.h in a CSPICE header file.

The translated routines' argument list and f2c internal string formats caused most of the problems during the first stage of CSPICE development. As mentioned, f2c converted code use FORTRAN strings, consequently a string passed from a standard C routine to a translated routine must be converted to a FORTRAN string; a string passed from a translated routine to a standard C routine must be converted to a C string. Translated routines do not detect possible error modes such as zero length strings or null character pointers, so several subroutines and a set of macros handle these string operations. As in other FORTRAN-C interfaces, the routines require string length arguments.

Performance

The C code created by f2c may well emulate the behavior of the source FORTRAN code, but that code tends to run slow without compiler optimization.

CSPICE uses optimization on all target platforms when possible. A small difficulty expressed itself during the first attempts to compile CSPICE under CodeWarrior Pro 3. If the compiler optimization settings are:

```
Instruction Scheduling for 604,
Optimize for SPEED,
Global Optimization Level 4,
Peephole Optimization
```

several routines fail to compile due to memory constraints on the CodeWarrior IDE. Trial and error proved the culprit to be Global Optimization Level 4, so the offensive routines now compile at Global Optimization Level 1. Another routine resists optimization on the MS-PC platform; this routine compiles unoptimized.

C++ issues

CSPICE is the basis for a prototype object oriented version of the SPICE library in C++ (SPICE++, still under design). Several problems were found with calls to CSPICE from C++ code.

All input arrays and matrices to wrappers are of type ConstSpiceDouble (const double). Some pesky compilers flag a "non-const passed to a const" warning when passing non-const arguments. Nat Bachman created a set of interface macros which perform type casting for the appropriate routines. The macros prevent the warning without forcing the user to explicitly declare their vectors or arrays as const.

Another issue is name mangling. A C++ compiler does not create the same symbol name for a routine as does a C compiler - a consequence of C++ function overload property. To link a C library to a C++ program requires that the C routine be defined as an external routine. Nat Bachman added a compile time flag to identify the CSPICE prototypes as external functions when linking the CSPICE library against a C++ routine.

Applications of the CSPICE Library.

Example 1 shows a simple program (the states cookbook program) with a complete header, which retrieves SPICE kernel data to calculate a body's state with respect to some observer in a user defined reference frame. The program demonstrates how to load SPK, PCK, and leapseconds kernel files, convert a time/date string to the epoch time (a time measured against a known reference date), then retrieve a state in a particular reference frame at the epoch time.

Other CSPICE uses:

IDL - CSPICE and a collection of interface routines allows IDL to access SPICE kernel data.

SSC - Solar System Calculator, a simple scripting interface to CSPICE. Originally designed by Mike Spencer, updated to use CSPICE by Ed Wright.

SOAP - a sophisticated orbit analysis tool by The Aerospace Corporation for the MacPPC, Windows NT and Solaris. SOAP details are available at:

http://www.aero.org/software/soap/index.html.

The use of CSPICE gives SOAP users the ability to visualize trajectory data and viewing geometry from data in SPICE kernels.

Future Work

CSPICE work continues for the foreseeable future. Current goals:
- expand the number of wrappers
- improve platform compatibility
- C specific documentation
- bug squashing

Long range goals:
- An object-oriented library (SPICE++)
- decoupling the f2c I/O libraries from CSPICE
- use of dynamic memory allocation in low level CSPICE routines
- complete documentation with tutorial code examples

CSPICE is not designed for multithreaded applications nor does NAIF does not plan to add this capability. SPICELIB and CSPICE are stable across all dates and calendars.

Alpha versions of the CSPICE toolkit are available via anonymous ftp at naif.jpl.nasa.gov in /pub/naif/cspice.

NASA supports CSPICE development under contract #?. My thanks to Nat Bachman who provided input and reviews of this article.

Example 1.

```
/*

-Procedure states( Compute state of one body relative to another )

-Abstract

   This "cookbook" program demonstrates the use of NAIF S- and P-
   Kernel (SPK) files and subroutines to calculate the state
   (position and velocity) of one solar system body relative to
   another solar system body.

   The purpose of this program is twofold:

      1) To show how NAIF ephemeris data may be made available to
         a program.

      2) To show how the apparent, true, or geometric state
         (inertially referenced cartesian position and velocity)
         of one solar system body relative to another solar
         system body may be calculated.

   The CSPICE subroutine spklef_c (S/P Kernel, Load Ephemeris
   File) handles the first task by maintaining a database of
   ephemeris files. The calling program indicates which files
   to load by passing their names to spklef_c.

   spkezr_c (S/P Kernel, Real easy reader) handles the second task
   by accessing the data loaded with spklef_c (spkezr_c does not
   require the name of an SPK file as input).

-Copyright

   Copyright (1998), California Institute of Technology.
   U.S. Government sponsorship acknowledged.

-Input

   The program prompts the user for the following input:

      - The name of a NAIF leapseconds kernel file.
      - The name of a NAIF binary SPK ephemeris file.
      - The name for the observing body.
```

- The name for the target body.
- A time string of interest.
- An reference frame, i.e., "J2000".
- The type of aberration correction desired.

-Output

- The state of the target body relative to the observing body.
- **The one-way light-time from the target body to the** observing body.

-Particulars

The user supplies a NAIF leapseconds kernel file, a NAIF binary SPK ephemeris file, valid names for both the target and observing bodies, and the time to calculate the body's state.

Note that the 'target body' and the 'observing body' are both NAIF ephemeris objects described via their common names, and may be any of the following, provided ephemeris data are available for them in the SPK file:

- a spacecraft
- a planet or satellite mass center
- a planet barycenter
- the sun
- the solar system barycenter
- a comet
- an asteroid

By definition, the ephemerides in SPK files are continuous. The user can obtain states for any epoch within the interval of coverage. Epochs are always specified in ephemeris seconds past Julian year 2000 when accessing SPK files.

The ephemeris data in a single SPK file may be referenced to a number of different (inertial or non-inertial) frames. The user can specify that states be returned in any of the recognized frames listed in the NAIF IDs Required Reading, including J2000 and B1950.

spkezr_c returns apparent, true, or geometric states depending on the value of the aberration flag when it is called.

Flag	Type of correction	State computed by spkezr_c
"LT+S"	light-time and stellar aberration	Apparent
"LT"	light-time only	True

```
   "NONE"   no correction                                Geometric

   For the sake of brevity, this program performs no error checks
   on its inputs. Mistakes will cause the program to crash.

-References

   For additional information, see NAIF IDS Required Reading, and
   the headers of the CSPICE subroutines spklef_c, spkez_c and
   str2et_c.

-Restrictions

   None.

-Literature_References

   None.

-Author_and_Institution

   E.D. Wright     (JPL)

-Version

   -CSPICE Version 1.0.0, 01-MAR-1998   (EDW)

-&
*/


   /* Load needed headers. */

   #include <stdio.h>
   #include "SpiceUsr.h"


   /* Local declarations. */

   #define                 UTCLEN   48
   #define                 LENOUT   32
   #define                 FILELEN  72

   SpiceDouble             vec     [3];
   SpiceDouble             vec1    [3];
   SpiceDouble             vec2    [3];
   SpiceDouble             vout    [3];
   SpiceDouble             state   [6];
   SpiceDouble             lt;
   SpiceDouble              et;

   SpiceChar              * leap;
   SpiceChar              * spk;
   SpiceChar              * corr;
   SpiceChar              * ref;
   SpiceChar              * utc;
   SpiceChar              * format;
   SpiceChar              * targ;
   SpiceChar              * obs;
   SpiceChar                utcstr[ UTCLEN ];

   SpiceInt                 prec;
   SpiceInt                 handle;


void main()
   {


   /*
   Set the time output format and the precision of that output.
   */
   format = "C";
   prec   = 0;


   /*
   Start out by prompting for the names of kernel files.
   Load each kernel as the name is supplied.

   Note:  prompt_c will allocate the needed memory for the
   returned strings.
   */


   /* Get and load the leapsecond kernel. */
   leap   = prompt_c ( "Enter name of leapseconds kernel    : ");
   ldpool_c ( leap );


   /* Get and load the spk kernel. */
   spk    = prompt_c ( "Enter name of SPK file              : ");
   spklef_c ( spk, &handle );


   /* Get the rest of the needed parameters. */
   targ   = prompt_c ( "Target (what am I looking at)       : ");
ref     = prompt_c ( "Reference frame (J2000, B1950, etc.): ");
corr    = prompt_c ( "Aberration correction                : ");
obs     = prompt_c ( "Observer (where am I)                : ");
utc     = prompt_c ( "Event time                           : ");


/* Convert the time string to ephemeris time J2000. */
str2et_c ( utc, &et );


/* Compute the state of targ from obs at et. */
spkezr_c (  targ, et, ref, corr, obs, state, &lt );


/* Convert the ephemeris time to a calendar format. */
et2utc_c ( et , format, prec, UTCLEN, utcstr );


/*
Everything's computed.  Output the results.  Units are
kilometers and kilometers per second.
*/
printf ("\n The state of %s wrt %s at UTC time %s\n", targ,
                                                        obs,
                                                        utcstr );
printf ( " X :  %f KM \n", state[0] );
printf ( " VX:  %f KMS\n", state[3] );
printf ( " Y :  %f KM \n", state[1] );
printf ( " VY:  %f KMS\n", state[4] );
printf ( " Z :  %f KM \n", state[2] );
printf ( " VZ:  %f KMS\n", state[5] );

}
```